Matrix: Multihop Address allocation and dynamic any-To-any Routing for 6LoWPAN

Bruno P. Santos[a,*], Otavio A. de O. Souza[a], Bruna Peres[a], Olga Goussevskaia[a], Marcos A. M. Vieira[a], Luiz F. M. Vieira[a], Antonio A. F. Loureiro[a]

[a]*Computer Science Department, Universidade Federal de Minas Gerais (UFMG) Av. Antonio Carlos 6627, Belo Horizonte, MG, Brazil.*

Abstract

Standard routing protocols for IPv6 over Low power Wireless Personal Area Networks (6LoWPAN) are mainly designed for data collection applications and work by establishing a tree-based network topology, enables packets to be sent upwards, from the leaves to the root, adapting to dynamics of low-power communication links. In this work[1], we propose Matrix, a platform-independent routing protocol that utilizes the existing tree structure of the network to enable reliable and efficient any-to-any data traffic in 6LoWPAN. Matrix uses hierarchical IPv6 address assignment to optimize routing table size, while preserving bidirectional routing. Moreover, it uses a local broadcast mechanism to forward messages to the right subtree when a persistent node or link failures occur. We implemented Matrix on TinyOS and evaluated its performance both analytically and through simulations on TOSSIM. Our results showed that the proposed protocol is superior to available protocols for 6LoWPAN when it comes to any-to-any data communication, concerning reliability, message efficiency, and memory footprint.

Keywords: 6LoWPAN, IPv6, CTP, RPL, any-to-any routing, fault tolerance

*Corresponding author
Email addresses: `bruno.ps@dcc.ufmg.br` (Bruno P. Santos), `oaugusto@dcc.ufmg.br` (Otavio A. de O. Souza), `bperes@dcc.ufmg.br` (Bruna Peres), `olga@dcc.ufmg.br` (Olga Goussevskaia), `mmvieira@dcc.ufmg.br` (Marcos A. M. Vieira), `lfvieira@dcc.ufmg.br` (Luiz F. M. Vieira), `loureiro@dcc.ufmg.br` (Antonio A. F. Loureiro)

[1]This manuscript is based on preliminary conference versions [1, 2].

1. Introduction

IPv6 over Low-power Wireless Personal Area Networks (6LoWPAN[2]) is a working group inspired by the idea that even the smallest low-power devices should be able to run the Internet Protocol to become part of the Internet of Things. Standard routing protocols for 6LowPAN, such as CTP (Collection Tree Protocol [3]) and RPL (IPv6 Routing Protocol for Low-Power and Lossy Networks [4]), have two distinctive characteristics: communication devices use unstructured IPv6 addresses that do not reflect the topology of the network (typically derived from their MAC addresses), and routing lacks support for any-to-any communication since it is based on distributed collection tree structures focused on bottom-up data flows (from the leaves to the root).

The advantage of a tree topology is a small routing table since each node just needs to establish who his parent-node is and maintain only that information for packet forwarding. The disadvantage of such structures is that other communication patterns, like downward (point-to-multipoint, P2MP) or any-to-any (peer-to-peer, P2P) bi-directional data flow, are not easily implemented. The problem with such one-directional routing is it makes it inefficient to build essential network functions, such as configuration routines and reliable mechanisms to ensure the delivery of end-to-end data. In order to do that, addition communication routines have to be implemented, and extensive routing information has to be inserted into the routing tables of memory-constrained nodes. Therefore, such standard protocols do not allow a straightforward implementation of any-to-any communication patterns.

Even though CTP does not support any-to-any traffic, the specification of RPL defines two modes of operation for top-down data flows: the non-storing mode, which uses source routing, and the storing mode, in which each node

[2]We use the acronym 6LoWPAN to refer to Low power Wireless Personal Area Networks that use IPv6

maintains a routing table for all possible destinations. This requires $O(n)$ space (where n is the total number of nodes), which is unfeasible for memory-constrained devices. Our experiments show that in random topologies with one hundred nodes, with no link or node failures, RPL succeeds to deliver less than 20% of top-down messages sent by the root (see Figure 9).

Some works have addressed this problem from different perspectives. In [5], ORPL, an extension of RPL, uses bloom filters and opportunistic routing to decrease control traffic overload. In [6], point-to-point communication is enabled due to each node in a collection tree stores the addresses of its direct and indirect child nodes using Bloom filters. In [7], XCTP is an extension of CTP, which uses opportunistic and reverse-path routing to enable bi-directional communication. XCTP is efficient concerning message overload, but exhibits the problem of high memory footprint, since each node needs to store an entry in the local routing table for every data flow going through that node.

In this work, we build upon the idea of using hierarchical IPv6 address allocation and propose Matrix, a routing scheme for dynamic network topologies and fault-tolerant any-to-any data flows in 6LoWPAN. Matrix assumes there is an underlying collection tree topology (provided by CTP or RPL, for instance), in which nodes have static locations, i.e., are not mobile, and links are dynamics, i.e., nodes might choose different parents according to link quality dynamics. Matrix uses only one-hop information in the routing tables and implements a local broadcast mechanism to forward messages to the right subtree when node or link failures occur. Local broadcast is activated by a node when it fails to forward a message to the next hop (subtree) in the address hierarchy.

After the network has been initialized and all nodes have received an IPv6 address range, three simultaneous distributed trees are maintained by all nodes: the collection tree (Ctree), the IPv6 address tree (IPtree), and the reverse collection tree (RCtree), reflecting the dynamics of the collection tree in the reverse direction. Initially, any-to-any packet forwarding is performed using Ctree for bottom-up, and IPtree for top-down data flows. Whenever a node or link fails or Ctree changes, the new link is added in the reverse direction into RCtree, and it remains as long as this topology change persists. Top-down data packets are then forwarded from IPtree to RCtree via a local broadcast. Whenever a node receives a local-broadcast message, it checks whether it knows the subtree of the destination IPv6 address: if yes then the node forwards the packet to the right subtree via RCtree and the packet continues its path in the IPtree until the final destination.

Why is this approach robust to network dynamics? Routing is performed using the address hierarchy represented by the IPtree, so whenever a link or node fails, messages addressed to destinations in the corresponding subtree may be lost. Matrix uses the (dynamic) reverse collection tree and the local broadcast mechanism to forward messages to the right subtree, as long as an alternative route exists. Note that this local rerouting mechanism does not guarantee that all messages will be delivered. We argue that the probability that the message will be forwarded to the appropriate subtree is high, as long as there is a valid path, due to the geometric properties of wireless networks. Our simulations showed that this intuition is, in fact, correct. In adverse network conditions, without the local broadcast mechanism, Matrix delivers 85% of top-down messages when a route exists; with the local broadcast activated, the success rate increases to 95% (roughly 2/3 of otherwise lost messages succeed in reaching the final destination).

Why does this approach scale? Each node stores only one-hop neighborhood information, namely: the id of its parent in Ctree, the IPv6 address ranges of its children in the IPtree, and the IPv6 address ranges of its (temporary) children in the RCtree. Therefore, the memory footprint at each node is $O(k)$, where k is the number of children at any given moment in time. The impact of such low memory footprint on the end-to-end routing success is impressive: whereas RPL delivers less than 20% of packets in some scenarios, Matrix delivers 99% of packets successfully, without end-to-end mechanisms.

We evaluated the proposed protocol both analytically and by simulation. Even though Matrix is platform-independent, we implemented it as a subroutine of CTP on TinyOS and conducted simulations on TOSSIM. The results showed that, when it comes to any-to-any communication, Matrix presents significant gains in terms of reliability (high any-to-any message delivery) and scalability (presenting a constant, as opposed to linear, memory complexity at each node) at a moderate cost of additional control messages, when compared to other state-of-the-art protocols, such as XCTP and RPL.

To sum up, Matrix achieves the following essential goals that motivated our work:

- **Any-to-any routing**: Matrix enables end-to-end connectivity between hosts located within or outside the 6LoWPAN.
- **Memory efficiency**: Matrix uses compact routing tables and, therefore, is scalable to extensive networks.
- **Reliability**: Matrix achieves 99% delivery without end-to-end mechanisms, and delivers $\geq 90\%$ of end-to-end packets when a route exists under challenging network conditions.
- **Communication efficiency**: Matrix uses adaptive beaconing based on Trickle algorithm [8] to minimize the number of control messages in dynamic network topologies (except with node mobility).
- **Hardware independence**: Matrix does not rely on specific radio chip features, and only assumes an underlying collection tree structure.
- **IoT integration**: Matrix allocates global (and structured) IPv6 addresses to all nodes, which allow nodes to act as destinations integrated into the Internet, contributing to the realization of the Internet of Things.

The rest of this paper is organized as follows. In Section 2, we describe the Matrix protocol design. In Section 3, we analyze the message complexity of the protocol. In Section 4, we present our analytic and simulation results. In Section 5, we discuss some related work. Finally, in Section 6, we present the concluding remarks.

2. Design overview

The objective of Matrix is to enable any-to-any routing in an underlying data collection protocol for 6LoWPAN, such as CTP and RPL, while preserving memory and message efficiency, as well as adaptability to networks topology dynamics[3]. Matrix is a network layer protocol that works together with a routing protocol. Figure 1 illustrates the protocol's architecture, which is divided into: *routing engine* and *forwarding engine*. The routing engine is responsible for the address space partitioning and distribution, as well as routing table maintenance. The forwarding engine is responsible for application packet forwarding.

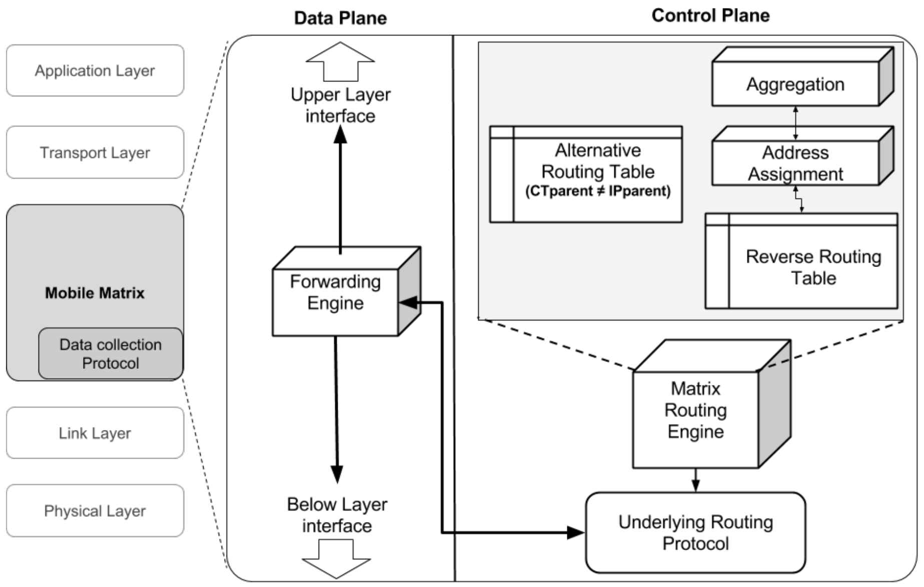


Figure 1: Matrix protocol's architecture.

Matrix encompasses of the following execution phases:

[3]Note that Matrix is not designed to address scenarios with node mobility, but only to work with network topology dynamics caused by changes in link quality, as well as node and link failures.

1. **Collection tree initialization**: the collection tree (Ctree) is built by the underlying collection protocol; each node achieves a stable knowledge about who its parent is; adaptive beaconing based on Trickle algorithm [8] is used to define stability;
2. **IPv6 multihop host configuration**: once the collection tree is stable, the address hierarchy tree (IPtree) is built using MHCL (Section 2.1); this phase also uses adaptive beaconing to handle network dynamics; by the end of this phase, each node has received an IPv6 address range from its parent, and each non-leaf node has partitioned its address space among its children; the resulting address hierarchy is stored in the distributed IPtree, which initially has the same topology as Ctree, but in reverse, top-down, direction.
3. **Standard routing**: bottom-up routing is done using the collection tree, Ctree, and top-down routing is done using the address hierarchy represented by the IPtree; any-to-any routing is performed by combining bottom-up forwarding, until the least common ancestor of sender and receiver, and then top-down forwarding until the destination.
4. **Alternative top-down routing table upkeep**: whenever a node changes its parent in the initial collection tree, it starts sending beacons to its new parent in Ctree, requesting to upkeep an entry in its routing table with its IPv6 range; such new links in Ctree, in reverse direction, comprise the RCtree routing tables for alternative (top-down) routing;
5. **Alternative top-down routing via local broadcast**: whenever a node fails to forward a data packet to the next hop/subtree in the IPtree, it broadcasts the packet to its one-hop neighborhood; upon receiving a local broadcast, all neighbors check if the destination IPv6 belongs to an address range in their RCtree table; if positive, the packet is forwarded to the correct subtree of IPtree. Otherwise, the packet is dropped; we give a geometric argument and show through simulations that such events are rare.

Next, we describe the architecture of Matrix in more detail.

2.1. MHCL: Multihop Host Configuration for 6LoWPAN

Matrix is built upon the idea of IPv6 hierarchical address allocation. The address space available to the border router of the 6LoWPAN (e.g., the 64 least-significant bits of the IPv6 address or a compressed 16-bit representation of it) is hierarchically partitioned among nodes connected to the border router through a multihop cycle-free topology (implemented by standard protocols, such as RPL or CTP). Each node receives an address range from its parent and partitions it among its children, until all nodes receive an address. Since the address allocation is performed hierarchically, the routing table of each node has k entries, where k is the number of its (direct) children. Each routing table entry aggregates the addresses of all nodes in the subtree rooted at the corresponding child-node. A portion, say $r\%$, of the address space available to each node is left reserved for possible future/delayed connections (parameter r can be configured according to the expected number of newly deployed nodes in the network, see Figure 2). We refer to the resulting distributed tree structure as IPtree.

Messages: MHCL uses two message types: $MHCL_A$ ($MHCL_{Aggregation}$) e $MHCL_D$ ($MHCL_{Distribution}$). Messages of type $MHCL_A$ are used in the upward routes, from child to parent. This message carries the number of a node's descendants, used in the aggregation phase. Messages of type $MHCL_D$ are sent along downward routes, from parent to child. This message is used for address allocation and contains the address and corresponding address partition assigned to a child node by its parent. Note that the size of the first address and the size of the allocated address partition can have a length predefined by the root, according to the overall address space (we used a value of 16 bits because the compressed host address has 16 bits). This information is sufficient for the child node to decode the message and execute the address allocation procedure for its children.

Network stabilization: In order to decide how the available address space is partitioned, nodes need to collect information about the topology of the network. Once a *stable* view of the network's topology is achieved, the root starts distributing address ranges downwards to all nodes. Note that the notion of stability is important to implement a coherent address space partition. Therefore, MHCL has an initial set-up phase, during which information about the topology is progressively updated until a (predefined) sufficiently long period goes by without any topology changes. To implement this adaptive approach, we use Trickle-inspired timers to trigger messages (Algorithm 1). In Algorithm 1 two parameters are used: $Trickle_{min}$ is the minimum time interval used by the Trickle algorithm, and *spChild* is a multiplication factor used to define the maximum time interval, such that, if no changes occur within it, then the parent choice becomes stable, and the local variable $parentDefined$ is set to TRUE. Note that, once the network reaches an initial state of stability, later changes to topology are expected to be of local nature, caused by a link or a node failure, or a change in the preferred parent of a node. In these cases, the address allocation does not need to be updated, since local mechanisms of message resubmission can be used to improve message delivery rates, as described in Section 2.4.

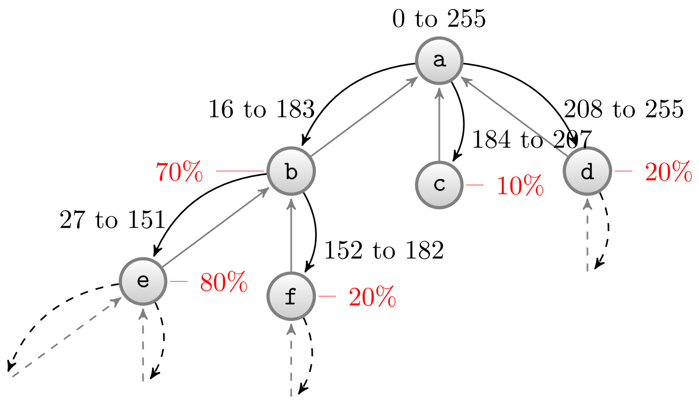


Figure 2: MHCL: simplified IPtree example: 8-bit address space at the root and 6.25% reserved for future/delayed connections.

Descendants convergecast: After the initial network stabilization, each node n_i counts the total number of its descendants, i.e., the size of the subtree

Algorithm 1 MHCL: Stabilization timer

```
1: parentDefined = FALSE;
2: maxTime = $spChild * Trickle_{min}$ ;
3: timer = rand(1/2 * $Trickle_{min}$, $Trickle_{min}$];                ▷ reset timer
4: while NOT parentDefined do
5:     if NOT-ROOT and TIMER-OFF then
6:         if PARENT-CHANGED then
7:             reset timer;
8:         else
9:             if $timer < maxTime$ then
10:                timer *= 2;                                        ▷ double timer
11:            else
12:                parentDefined = TRUE;
13:            end if
14:        end if
15:    end if
16: end while
```

rooted at itself, and propagates it to its parent. Moreover, n_i saves the number of descendants of each child. If a node is not the root, and it has defined who the preferred parent is (*parentDefined* is TRUE) it starts by sending a $MHCL_A$ message with *count* = 0 (Algorithm 2). Then it waits for $MHCL_A$ messages from its children, updates the number of descendants of each child, and propagates the updated counter to the parent until its total number of descendants is stable. If a node is the root, then it just updates the number of descendants of each child by receiving $MHCL_A$ messages until its total number of descendants is stable (Algorithm 3). Parameters *spLeaf* and *spRoot* are used to define stabilization criteria in non-root nodes and the root node, respectively. Once the aggregation phase is completed, the root's local variable *descendantsDefined* is set to TRUE.

Address allocation: Once the root has received the (aggregate) num-

Algorithm 2 MHCL: Aggregation timer (non-root nodes)

```
1:  maxTimeLeaf = $spLeaf * Trickle_{min}$;
2:  timer = rand(1/2 * $Trickle_{min}$, $Trickle_{min}$];                ▷ reset timer
3:  count = 0;                  ▷ counts descendants through $MHCL_A$ messages
4:  while NO-$MHCL_D$-FROM-PARENT do
5:                                                  ▷ hasn't received IPv6 range
6:      if NOT-ROOT and TIMER-OFF then
7:          if parentDefined and ($count < 1$) then
8:              send $MHCL_A$ to parent;                    ▷ trigger aggregation
9:          end if
10:         if COUNT-CHANGED then
11:             send $MHCL_A$ to parent;
12:             reset timer;
13:         else
14:             if $timer < maxTimeLeaf$ then
15:                 timer *= 2;
16:             end if
17:         end if
18:     end if
19: end while
```

ber of descendants of each child; it splits the available address space into k ranges proportionally to the size of the subtree rooted at each child (see Algorithm 4). Each node n_i repeats the space partitioning procedure upon receiving its address space from the parent and sends the proportional address ranges to the respective children (always reserving $r\%$ for delayed address allocation requests). The idea is to allocate larger portions to larger subtrees, which becomes important in especially large networks because it maximizes the address space utilization. Note that this approach needs information aggregated along multiple hops, which results in a longer set-up phase.

Delayed connections: If an address allocation request from a new child

Algorithm 3 MHCL: Aggregation timer (Root)

```
1: descendantsDefined = FALSE;
2: maxTimeRoot = $spRoot * Trickle_{min}$;
3: timer = rand($1/2 * Trickle_{min}$, $Trickle_{min}$];        ▷ reset timer
4: count = 0;        ▷ counts descendants through $MHCL_A$ messages
5: while NOT descendantsDefined do
6:     if IS-ROOT and TIMER-OFF then
7:         if COUNT-CHANGED then
8:             reset timer;
9:         else
10:            if $timer < maxTimeRoot$ then
11:                $timer* = 2$;
12:            else
13:                descendantsDefined = TRUE;
14:            end if
15:        end if
16:    end if
17: end while
```

node is received after the address space had already been partitioned and assigned, then the address allocation procedure is repeated using the reserved address space.

After the address allocation is complete, each (non-leaf) node stores a routing table for downward traffic, with an entry for each child. Each table entry contains the final address of the address range allocated to the corresponding child, and all table entries are sorted in increasing order of the final address of each range. In this way, message forwarding can be performed in (sub)linear time.

Algorithm 4 MHCL: IPv6 address distribution

```
1: STABLE = descendantsDefined or NOT-ROOT;
2: if STABLE and (IS-ROOT or RECEIVED-$MHCL_D$) then
3:     partition available address space;
4:     for each child $c_i$ do
5:         send $MHCL_D$ to $c_i$;                ▷ send IPv6 "range"
6:         if NO $ack$ then
7:             send $MHCL_D$ to $c_i$;            ▷ retransmit
8:         end if
9:     end for
10: end if
```

2.2. Control plane: distributed tree structures

After the network is initialized and all nodes have received an IPv6 address range, three simultaneous distributed trees are maintained on all nodes in the 6LoWPAN: **Ctree:** the collection tree, maintained by the underlying collection protocol (CTP/RPL). **IPtree:** the IPv6 address tree, built during the network initialization phase and kept static afterward, except when new nodes join the network, in which case they receive an IPv6 range from the reserved space of the respective parent node in the collection tree.

RCtree: the reverse collection tree, reflecting the dynamics of the collection tree in the opposite direction.

Initially, IPtree has the same topology as the reverse-collection tree $Ctree^R$, and RCtree has no links (see Figure 3(a) and 3(b)).

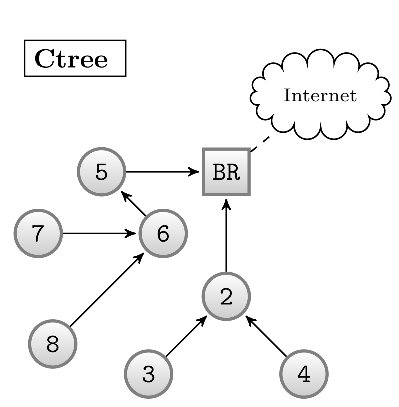
$$IPtree = Ctree^R \text{ and } RCtree = \emptyset$$

Whenever a change occurs in one of the links in Ctree, the new link is added in the reverse direction into RCtree and maintained as long as this topology change persists (see Figures 3(c) and 3(d)).

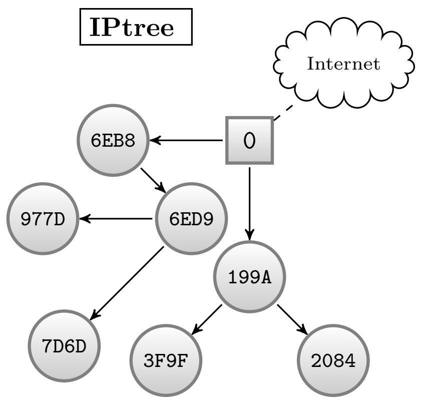
$$RCtree = Ctree^R \setminus IPtree$$

Therefore, RCtree is not really a tree since it contains only the reversed links present in Ctree but not in IPtree. Nevertheless, its union with the “working” links in IPtree is, in fact, a tree, which is used in the alternative top-down routing:

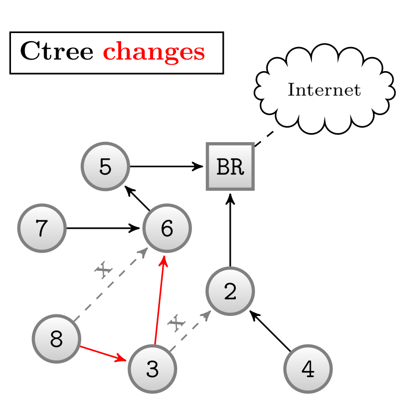
$$RCtree \cup (IPtree \cap Ctree^R) \text{ :alternative routing tree.}$$



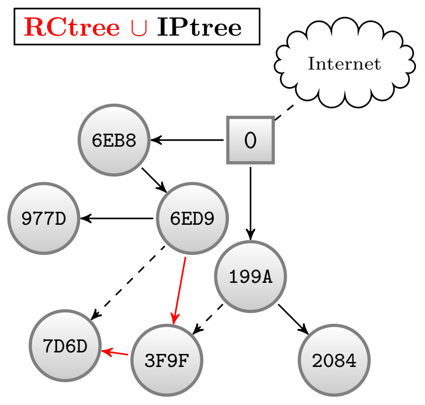
(a) Ctree structure, ID inside the nodes.



(b) IP addres assignment by MATRIX hierarchical distribution. Simplified 8-bit IP inside the nodes.



(c) Links failure cause topology changes.



(d) RCtree adapts upon topology changes.

Figure 3: RCtree example: before and after two links change in the collection tree.

Each node n_i maintains the following information:

- $CTparent_i$: the ID of the current parent in the dynamic collection tree;

- $IParent_i$: the ID of the node that assigned n_i its IPv6 range initially $CTarent_i = IParent_i$);

- $IPchildren_i$: the *standard* (top-down) routing table, with address ranges of each one-hop descendant of n_i in the IPtree;

- $RChildren_i$: the *alternative* (top-down) routing table, with address ranges of one-hop descendants in the RCtree.

Note that, each node stores only one-hop neighborhood information, so the memory footprint is $O(k)$, where k is the number of a node's children at any given moment in time, which is optimal, considering that any (optimal) top-down routing mechanism would need at least one routing entry for every (current) child in the tree topology to reach all destinations.

The routing engine (see Figure 1) is responsible for creating and maintaining the IPtree and RCtree routing tables. IPtree is created during the network initialization phase, while RCtree is updated dynamically to reflect changes in the network's link qualities. Whenever a node n_i has its $CTparent_i$ updated, and the current parent is different from its $IParent_i$ ($IParent_i \neq CTparent_i$), n_i starts sending periodic beacons to its new parent, with regular intervals (in our experiments, we set the beacon interval to $\delta/8$, where δ is the maximum interval of the Trickle timer used in CTP). Upon receiving a beacon (from a new child in the collection tree), a node ($n_j = CTparent_i$) creates and keeps an entry in its alternative routing table $RChildren_j$ with the IPv6 address range of the subtree of n_i. As soon as n_i stops using n_j as the preferred parent, it stops sending beacons to n_j. If no beacon is received from n_i after $2 \times \delta$ ms, its (alternative) routing entry is deleted. Therefore, links in RCtree are temporary and are deleted when not present in neither the collection nor the IP trees.

2.3. Data plane: any-to-any routing

The forwarding engine (see Figure 1) is responsible for application packet forwarding. Any-to-any routing is performed by combining bottom-up forwarding, until the least common ancestor of sender and receiver, and then top-down forwarding until the destination. Upon receiving an application layer packet, each node n_i verifies whether the destination IPv6 address falls within some range $j \in IPchildren_i$: if yes then the packet is forwarded (downwards) to node n_j, otherwise, the packet is forwarded (upwards) to $CTparent_i$. Note that, since each node has an IPv6 address, in contrast to collection protocols, such as CTP and RPL, in Matrix, every node can act as a destination of messages originated inside and outside of the 6LoWPAN.

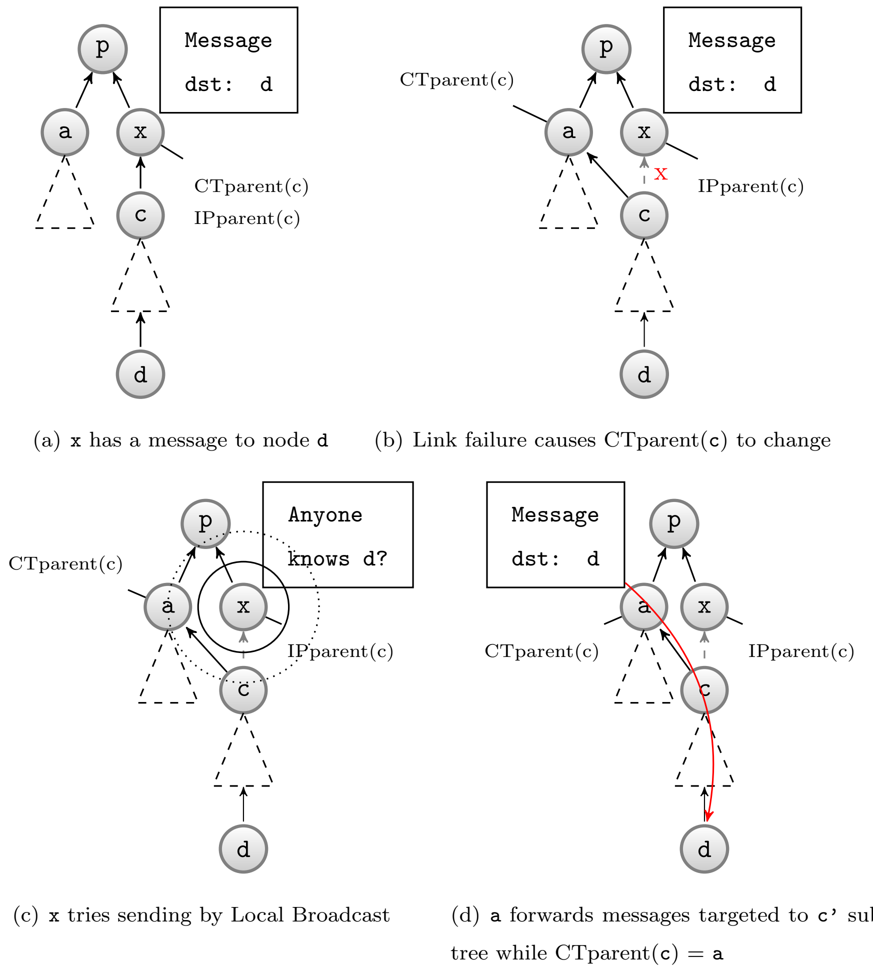
Each forwarded packet requests an acknowledgment from the next hop and can be retransmitted up to 30 times (similarly to what is done in CTP [3]). If after that no acknowledgment is received, then the node performs a *local broadcast*, looking for an alternative next hop in the RCtree table of a (one-hop) neighbor. The *alternative routing* process is described in detail below.

2.4. Fault tolerance and network dynamics

So why is Matrix robust to network dynamics? Note that, since routing is based on the hierarchical address allocation, if a node with the routing entries necessary to locate the next subtree becomes unreachable for longer than approximately one second (failures that last less than 1s are effectively dealt with by retransmission mechanisms available in standard link layer protocols), messages with destinations in that subtree are dropped.

When a node or link fails or changes in Ctree, RCtree reflects this change, and packets are forwarded from IPtree to RCtree via a local broadcast. The node that receives a local-broadcast checks in its RCtree whether it knows the subtree of the destination IPv6 address: if yes then is forwards the packet to the right subtree and the packet continues its path in the IPtree until the final destination.

Consider the following scenario: node X receives a packet with destination IPv6 address D (see Figure 4(a)). After consulting its standard routing table $IP - children_X$, X forwards the packet to C. However, the link X $\Rightarrow$ C fails, for some reason, and C does not reply with an acknowledgment. Then, X



(a) `x` has a message to node `d`

(b) Link failure causes CTparent(`c`) to change

(c) `x` tries sending by Local Broadcast

(d) `a` forwards messages targeted to `c`' subtree while CTparent(`c`) = `a`

Figure 4: Alternative top-down routing upon Ctree change.

makes a constant number (e.g., 30 times in CTP) of retransmission attempts. Meanwhile, since node C also lost its connection to X, it decides to change its parent in the collection tree to node A (see Figure 4(b)). Having changed its parent, C starts sending beacons to A, which creates an entry in its alternative routing table $RC - children_A$ for the subtree rooted at C, and keeps it as long as it receives periodic beacons from C (which will be done as long as $CTparent_C$ = A).

Having received no acknowledgment from C, X activates the *local broadcast* mode: it sets the message's type to "LB" and broadcasts it to all its one-hop neighbors (see Figure 4(c)). Upon receiving the local broadcast, node A consults its alternative routing table and finds out that the destination address D falls within the IPv6 address range C. It then forwards the packet to C, from where the packets follow along its standard route in the subtree of C (see Figure 4(d)).

Note that this mechanism does not guarantee that the message will be delivered. If no one-hop neighbor of X had the address range of C in its alternative routing table, then the packet would be lost. Nevertheless, we argue that the probability that the message will be forwarded to the appropriate subtree is high.

2.5. Alternative routing: geometric rationale

The success of the local broadcast mechanism lies in the ability to forward messages top down along the IPtree, in spite of one or more link or node failures on the way. Note that, whenever a node of IPtree is unavailable, it might not be possible to find the right subtree of the destination. Matrix is designed to handle (non-adjacent) link or node failures and relies on a single local broadcast and temporary reverse collection links (RCtree).

Consider once again the scenario illustrated in Figure 4. When a node X is unable to forward a packet to the next hop, it activates the local broadcast mechanism, and it becomes essential that one of X's one-hop neighbors (in this case A) has replaced X as a parent of C in the collection tree. Therefore, given that the new parent of C is A, it becomes essential that X and A are neighbors. We argue that it is unlikely that this is not the case.

Our argument is of geometric nature. Since the considered 6LoWPAN is wireless, we show our argument in a unit disk graph (UDG) model [9]. We use the fact that the number of independent neighbors of any node in a UDG is bounded by a small constant, namely 5. The proof of this fact is sketched in Figure 5: consider a node X and its neighbor A. Any node located inside the gray region is a neighbor of both X and A, so any neighbor of X that is

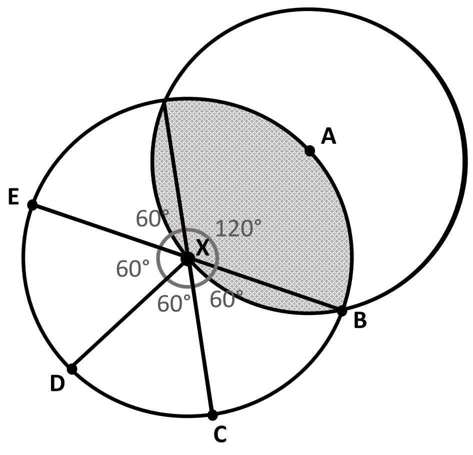


Figure 5: UDG model: the number of independent neighbors of X is at most 5.

independent of (not adjacent to) A has to be outside the gray area and inside the circle around X. Let's call this neighbor B. The next independent neighbor of X has to be located outside the 60-degree sector that starts at B, and so on. This procedure can be repeated no more than 5 times before the 360 degrees around X are covered.

Given that the maximum number of neighbors that do not know each other is very small, for any possible node distribution and density around X, the probability that two neighbors of X are independent is low. In Figure 4(c), since both X and A are neighbors of C, the probability that they are themselves neighbors is high. Similar arguments can be used to back the effectiveness of the local broadcast mechanism when dealing with different non-adjacent link and node failures.

Note that this reasoning is only valid in an open space without obstacles and, even then, does not guarantee that the message will be delivered. Nevertheless, our experiments show that this intuition is in fact correct, and Matrix has a 95%–99% message delivery success in scenarios with node failures of increasing frequency and duration.

3. Complexity Analysis

In this section, we assume a synchronous communication model with point-to-point message passing. In this model, all nodes start executing the algorithm simultaneously and time is divided into synchronous rounds, i.e., when a message is sent from node v to its neighbor u at time-slot t, it must arrive at u before time-slot $t + 1$.

We first analyze the message and time complexity of the IPv6 address allocation phase of Matrix. Then, we look into the message complexity of the control plane of Matrix after the network initialization phase.

Note that Matrix requires that an underlying acyclic topology (Ctree) has been constructed by the network before the address allocation starts, i.e., every node knows who its parent in the Ctree is. Moreover, one of the building blocks of Matrix is the address allocation phase, described in Section 2.1.

Theorem 1. *For any network of size n with a spanning collection tree Ctree rooted at node root, the message and time complexity of Matrix protocol in the address allocation phase is Msg($Matrix^{IP}$ $(Ctree, root)$) $= O(n)$ and Time ($Matrix^{IP}$ $(T, root)$) $= O(depth(Ctree))$, respectively. This message and time complexity is asymptotically optimal.*

Proof. The address allocation phase is comprised of a tree broadcast and a tree convergecast. In the broadcast operation, a message (with address allocation information) must be sent to every node by the respective parent, which needs $\Omega(n)$ messages. Moreover, the message sent by the root must reach every node at $depth(Ctree)$ hops away, which needs $\Omega(depth(Ctree))$ time-slots. Similarly, in the convergecast operation, every node must send a message to its parent after having received a message from its children, which needs $\Omega(n)$ messages. Also, a message sent by every leaf node must reach the root, at distance $\leq depth(Ctree)$, which needs $\Omega(depth(Ctree))$ time-slots. $\square$

Next, we examine the communication cost of the routines involved in the alternative routing, performed in the presence of persistent node and link failures.

Theorem 2. *Consider a network with n nodes and a failure event that causes $\mathcal{L}_{\mathcal{CT}}$ links to change in the collection tree Ctree for at most Δ ms. Moreover, consider a beacon interval of δ ms. The control message complexity of Matrix to perform alternative routing is $Msg(Matrix^{RC}) = O(n)$.*

Proof. Consider the $\mathcal{L}_{\mathcal{CT}}$ link changes in the collection tree Ctree. Note that $\mathcal{L}_{\mathcal{CT}} = O(n)$ since Ctree is acyclic and, therefore, has at most $n-1$ links. Every link that was changed must be inserted in the RCtree table of the respective (new) parent and kept during the interval Δ using regularly sent beacons from the child to the parent. Given a beacon interval of δ, the total number of control messages is bounded by $\Delta/\delta \times \mathcal{L}_{\mathcal{CT}} = O(n)$. □

Note that, in reality, the assumptions of synchrony and point-to-point message delivery do not hold in a 6LoWPAN. The moment in which each node joins the tree varies from node to node, such that nodes closer to the root tend to start executing the address allocation protocol earlier than nodes farther away from the root. Moreover, collisions, node, and link failures can cause delays and prevent messages from being delivered. We analyze the performance of Matrix in an asynchronous model with collisions and transient node and link failures of variable duration through simulations in Section 4.

4. Evaluation

In this section, we evaluate the performance of Matrix through simulations. In Section 4.1 we describe the simulations setup and used parameters. In Section 4.2 we evaluate the number of beacons transmitted, the usage of routing tables and memory footprint, the downward routing success and response rate success and compare Matrix with three state-of-the-art protocols: RPL [4], CTP [3] and XCTP [7].

4.1. Simulation setup

Matrix was implemented as a subroutine of CTP in TinyOS [10] and the experiments were run using the TOSSIM simulator [11]. We compare Matrix with and without the local broadcast mechanism, to which we refer as MHCL. RPL was implemented in Contiki [12] and was simulated on Cooja [13]. Table 1 lists the default simulation parameters used for each protocol, in a non-faulty scenario. We use the *LinkLayerModel* tool from TinyOS to generate the topology and connectivity model.

Firstly, we run the protocols over a static network scenario without link or node failures. Also, we simulated a range of faulty scenarios, based on experimental data collected from TelosB sensor motes, deployed in an outdoor environment [14]. In each scenario, after every 60 seconds of simulation, each node shutdowns its radio with probability σ and keeps the radio off for a time interval uniformly distributed in $[\varepsilon - 5, \varepsilon + 5]$ seconds (see Table 2). So, each scenarios represent a combination of values of σ and ε. Note that these are all node-failure scenarios, which are significantly harsher than models that simulate link or per-packet failures only.

On top of the network layer, we ran two different applications: top-down and any-to-any. In the top-down application, each node sends 10 messages to the root and the root replies with an acknowledgment. In the any-to-any application, each node chooses randomly 10 destination addresses and sends one message to each of those addresses. Nodes start sending application messages 90 seconds after the simulation has started. The entire simulation takes 20 minutes. Each simulation was run 10 times. In each plot, the curve or bars represent the average, and the error bars the confidence interval of 95%. For each protocol, only results relevant to each plot were included: e.g., CTP does not have a reverse routing table to performs top-down routing, and MHCL differs from Matrix only in faulty scenarios; otherwise, it performs equally and therefore was omitted.

Table 1: Simulation parameters

Parameter	Value
Base Station	1 center
Number of Nodes	100
Radio Range (m)	100
Density ($nodes/m^2$)	10
Number of experiments	10
Path Loss Exponent	4.7
Power decay (dB)	55.4
Shadowing Std Dev (dB)	3.2
Simulation duration	20 min
Application messages	10 per node
Max. Routing table size	20 entries

Table 2: Faulty network scenarios

Probability (σ) \ Duration (ε)	**Short Dur.**	**Moderate Dur.**	**Long Dur.**
Low Prob.	(1 %, 10 s)	(1 %, 20 s)	(1 %, 40 s)
Moderate Prob.	(5 %, 10 s)	(5 %, 20 s)	(5 %, 40 s)
High Prob.	(10 %, 10 s)	(10 %, 20 s)	(10 %, 40 s)

4.2. Results

Firstly, we turn our attention to memory efficiency of each protocol. To evaluate the use of routing tables, we compare the number of entries utilized by each protocol. Each node was allocated a routing table of maximum size equal to 20 entries. In Figure 6, we show the CDFs (cumulative distribution functions) of the percentage of routing table usage among nodes[4] for Matrix, RPL, XCTP, and MHCL.

[4]We measured the routing table usage of each node in one-minute intervals, then took the average over 20 minutes.

In this plot, Matrix was simulated in the faulty scenario, where σ and ε were setted to High Probability and Long Duration, respectivally (Table 2). Note that $> 35\%$ of nodes are leaves, i.e., do not have any descendants in the collection tree topology, and therefore use zero routing table entries.

As we can see, RPL is the only protocol that uses 100% of table entries for some nodes ($\geq 30\%$ of nodes have their tables full). This is because RPL, in the storing mode, pro-actively maintains an entry in the routing table of every node on the path from the root to each destination, which quickly fills the available memory and forces packets to be dropped.

XCTP reactively stores reverse routes only when required. Therefore, the number of routing entries used by XCTP depends on the number of data flows going through each node. Since the simulated flows were widely spaced during the simulation time, the XCTP was able to perform efficiently.

The difference between MHCL and Matrix is small: MHCL stores only the IPtree structure, whereas Matrix stores IPtree and RCtree data; the latter are kept only temporarily during parent changes in the collection tree, so its average memory usage is low.

Figure 7 illustrates the amount of control traffic in our experiments (the total number of beacons sent during the entire simulation). Matrix sends fewer control packet than RPL, because it only sends additional beacons during network initialization and in case of collection tree topology updates, whereas RPL has a communication intensive maintenance of downward routes during the entire execution time. Since XCTP is a reactive protocol, it does not send additional control packets, when compared to CTP.

Figure 8 compares RAM and ROM footprints in the protocol stack of CTP, RPL, XCTP, and Matrix. We can see that Matrix adds only a little more than 7KB of code to CTP, allowing this protocol to perform any-to-any communication with high scalability. When compared with RPL, the execution code of Matrix requires less RAM. Compared to XCTP, Matrix uses almost the same amount of RAM.

In Figure 9 we compare top-down routing success rate. We measured the

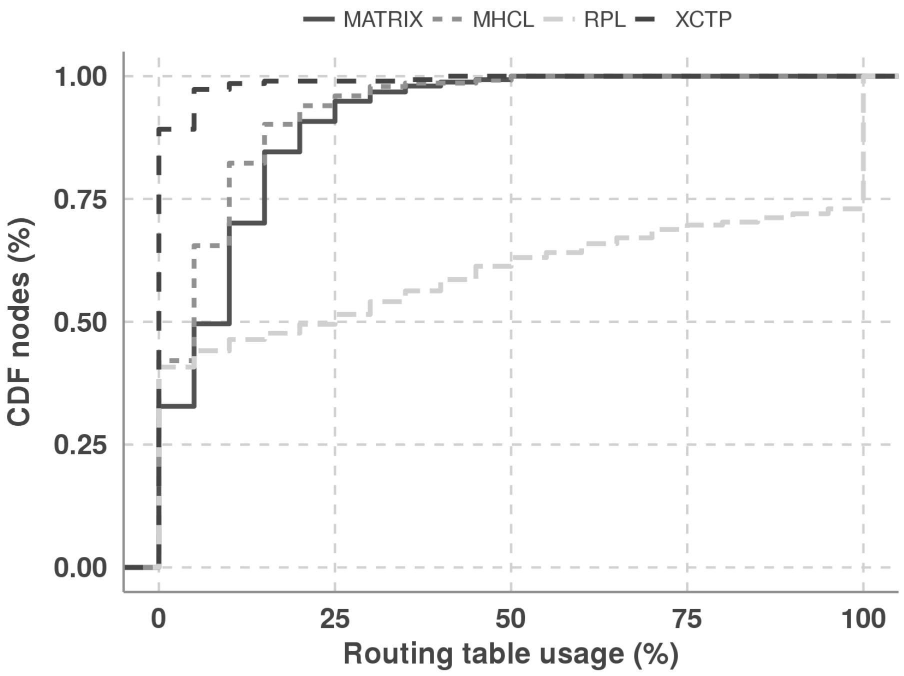
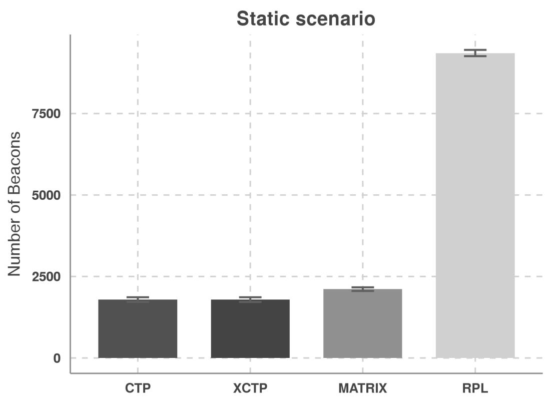


Figure 6: Routing table usage CDF. (Maximum table size = 20)

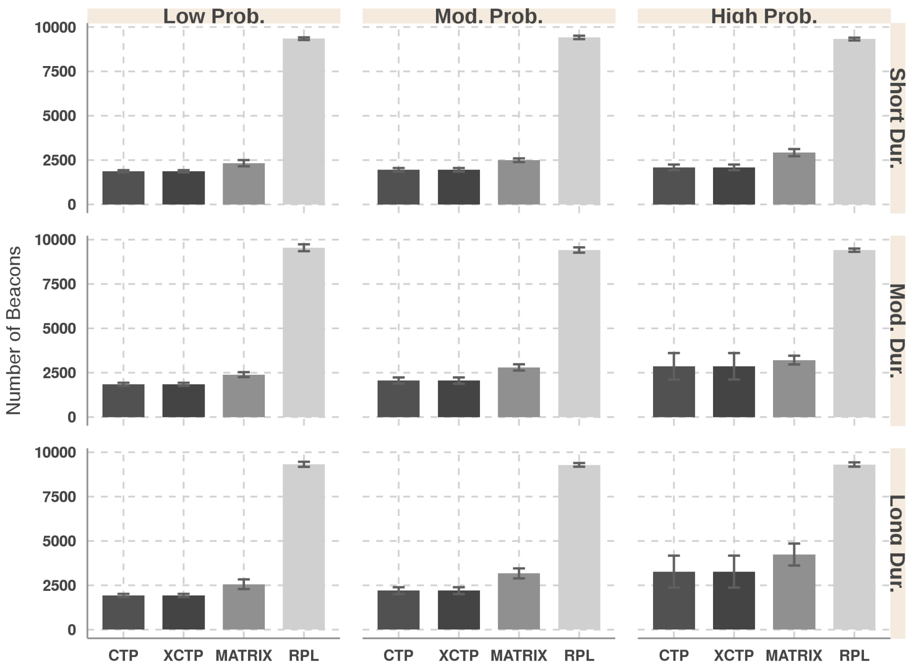
total number of application (ack) messages sent downwards and successfully received by the destination.[5] In the plot, “inevitable losses” refers to the number of messages that were lost due to a failure of the destination node, in which case, there was no valid path to the destination and the packet loss was inevitable. The remaining messages were lost due to wireless collisions and node failures on the packet’s path.

We can see that, when a valid path exists to the destination, the top-down success rate of Matrix varies between 95% and 99%. In the harshest faulty scenario 10, without the local broadcast mechanism, MHCL delivers 85% of top-down messages. With the local broadcast activated, the success rate in-

[5]We do not plot the success rate of bottom-up traffic, since it is done by the underlying collection protocol, without any intervention from Matrix.



(a) Static



(b) Faulty

Figure 7: Number of control packets.

creases to 95%, i.e., roughly 2/3 of otherwise lost messages succeed in reaching the final destination. Note that external factors may be causing RPL's low

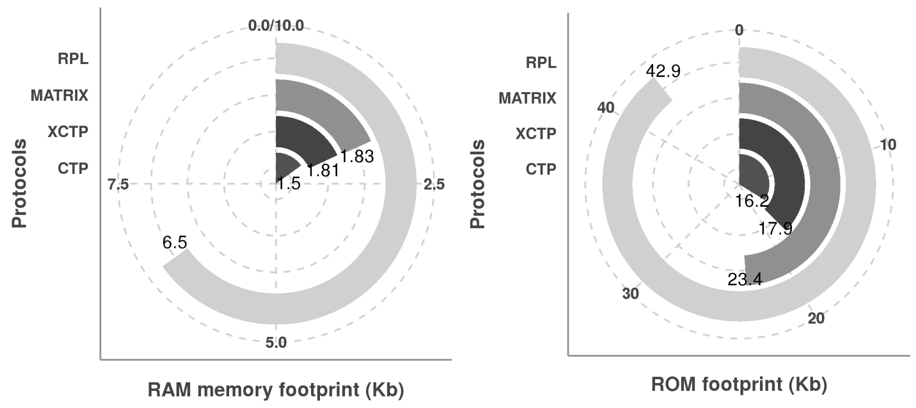
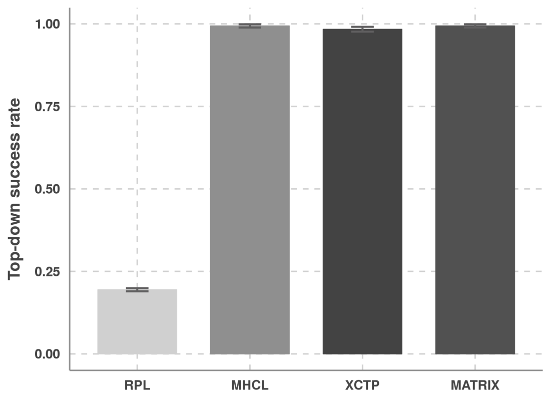


Figure 8: Code and memory footprint in bytes.

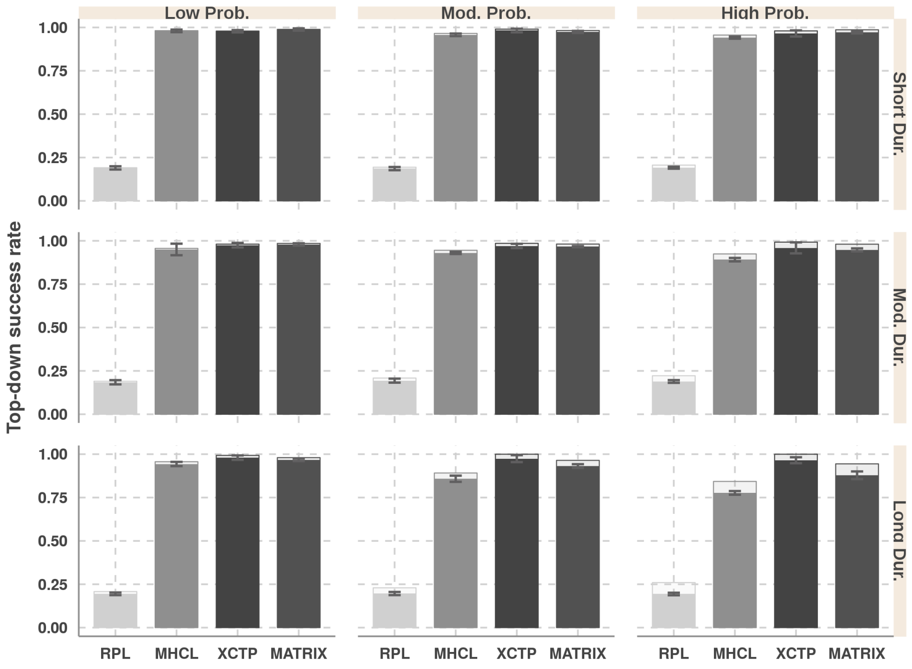
success rate. Since RPL was the only protocol implemented on Contiki and evaluated in Cooja, native protocols from this OS can interfere on the results. In [15], the authors show how different radio duty cycling mechanisms affect the performance of a RPL network. However, RPL delivered less than 20% of messages in all simulated scenarios, which occurs due to lack of memory to store all the top-down routes. Since XCTP is a reactive protocol, it adapts best to failures and dynamics, because downward routes are updated when a message travels upwards. In this way, the top-down success rate of XCTP is higher even in the presence of failures.

In Figure 10 we compare the any-to-any success rate. We measured the total number of messages sent by a node that was successfully received by the destination. In this application, each node chooses randomly a destination address and sends a message to this node. We can see that, as expected, there is no significant difference between any-to-any and top-down traffic patterns. Matrix performs any-to-any routing with 90% to 100% success rate, when a valid path exists to the destination. The success rate of RPL remains low, due to lack of memory to store all the routing information needed.

Finally, in Figure 11 we compare the response rate of Matrix and XCTP.



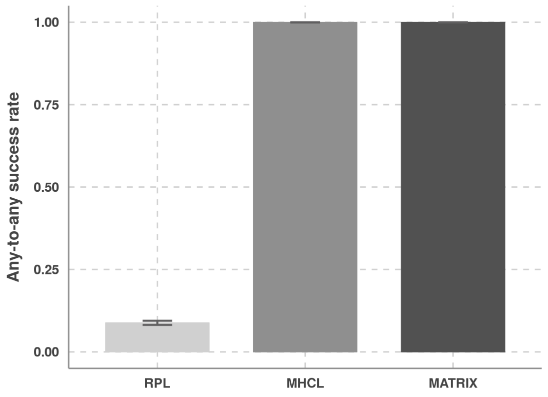
(a) Static



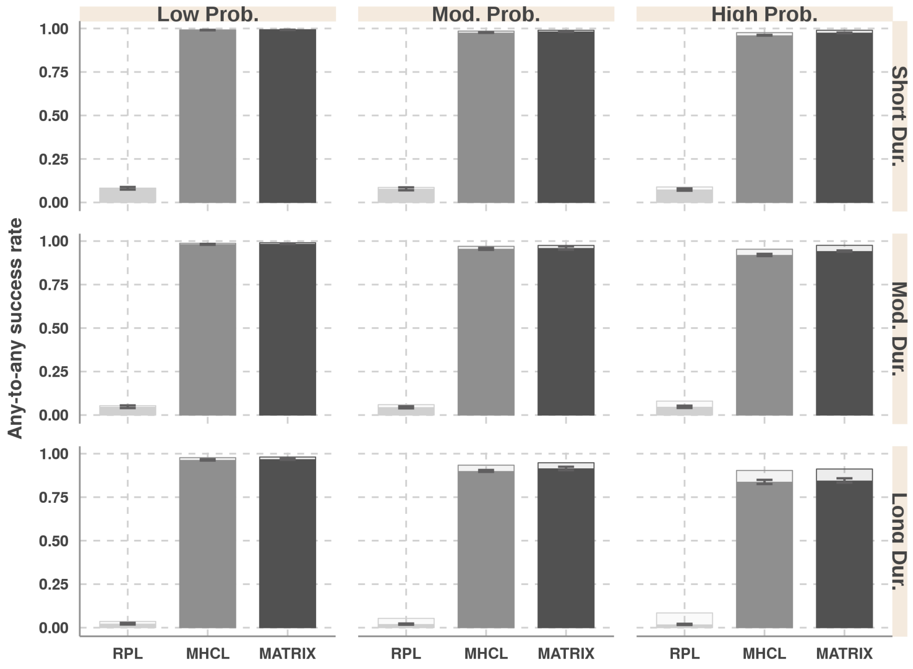
(b) Faulty

Figure 9: Top-down routing success rate.

We calculate the rate of reply by dividing the number of acknowledgments sent by the root by the number of messages received by the root. We vary the reply



(a) Static



(b) Faulty

Figure 10: Any-to-any routing success rate.

515 delay, that is, upon receipt of a message, the root will reply with an acknowledgment after x milliseconds, $x \in \{100, 200, 225, 250, 275, 300, 325, 350, 375, 400, 800\}$.

We can see that the performance of XCTP is highly dependent on the number of data flows. By increasing the application response delay, the number of simultaneous flows increases and the response success rate decreases, because nodes can not store all the information needed. Matrix, on the other hand, does not depend on the number of flows, and the routing table usage is bounded by the number of children of each node.

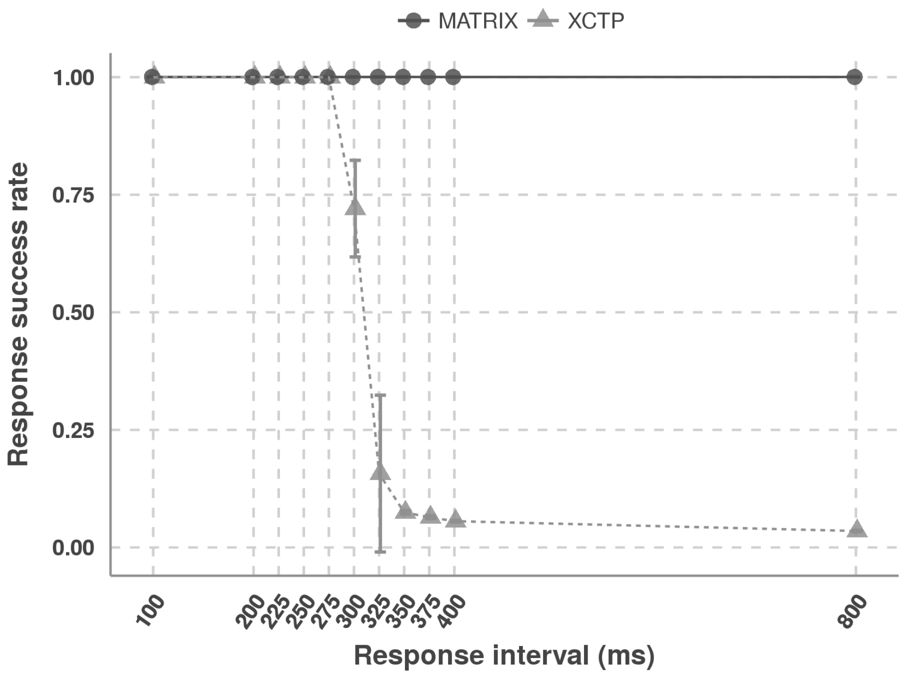


Figure 11: Response success rate.

5. Related Work

AODV [16] and DSR [17] are traditional wireless protocols that allow any-to-any communication, but they were designed for 802.11 and require too many states or apply several overheads on the packet header. In the context of low-power and lossy networks, CTP [3] and CodeDrip [18] were designed for bottom-up and top-down data flow, respectively. They support communication in only one direction.

State-of-the-art routing protocols for 6lowPAN that enable any-to-any communication are RPL [4], XCTP [7], and Hydro [19]. RPL allows two modes of operation (storing and non-storing) for downwards data flow. The non-storing mode is based on source routing, and the storing mode pro-actively maintains an entry in the routing table of every node on the path from the root to each destination, which is not scalable to even moderate-size networks. XCTP is an extension of CTP and is based on a reactive reverse collection route creating between the root and every source node. An entry in the reverse-route table is kept for every data flow at each node on the path between the source and the destination, which is also not scalable in terms of memory footprint. Hydro protocol, like RPL, is based on a DAG (directed acyclic graph) for bottom-up communication. Source nodes need to periodically send reports to the border router, which builds a global view (typically incomplete) of the network topology. Table 3 shows a comparison between the cited protocols for 6LoWPANs.

Some more recent protocols [20, 21, 22] modified RPL to include new features. In [20], a load-balance technique is applied over nodes to decrease power consumption. In [21, 22], they provide multipath routing protocols to improve throughput and fault tolerance.

Table 3: Comparison between related protocols for 6LoWPAN.

	MATRIX	**RPL**	**CTP**	**XCTP**
Bottom-up traffic	✓	✓	✓	✓
Top-down traffic	✓	✓		✓
Any-to-any traffic	✓	✓		
Address-allocation	✓			
Memory efficiency	✓		✓	
Fault tolerance	✓			✓

Acknowledgements

This work was supported in part by CAPES, CNPq and FAPEMIG.

6. Conclusions

In this paper, we proposed Matrix: a novel routing protocol that runs upon a distributed acyclic directed graph structure and is comprised of two main phases: (1) network initialization, in which hierarchical IPv6 addresses, which reflect the topology of the underlying wireless network, are assigned to nodes in a multihop way; and (2) reliable any-to-any communication, which enables message and memory-efficient implementation of a wide range of new applications for 6LoWPAN.

Matrix differs from previous work by providing a reliable and scalable solution for any-to-any routing in 6LoWLAN, both in terms of routing table size and control message overhead. Moreover, it allocates global and structured IPv6 addresses to all nodes, which allow nodes to act as destinations integrated into the Internet, contributing to the realization of the *Internet of Things*.

An interesting future direction is to study mobility in 6LoWPAN. We would like to evaluate the suitability of Matrix in mobile scenarios, where nodes change their point-of-attachment to the 6LowPAN without changing their IPv6 address, exploring features of the Mobile IPv6 protocol [23].

References

[1] B. S. Peres, O. A. d. O. Souza, B. P. Santos, E. R. A. Junior, O. Goussevskaia, M. A. M. Vieira, L. F. M. Vieira, A. A. F. Loureiro, Matrix: Multihop address allocation and dynamic any-to-any routing for 6lowpan, in: Proceedings of the 19th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems, MSWiM '16, ACM, New York, NY, USA, 2016, pp. 302–309. doi:10.1145/2988287.2989139.

[2] B. S. Peres, O. Goussevskaia, Ipv6 multihop host configuration for low-power wireless networks, in: Computer Networks and Distributed Systems (SBRC), 2015 XXXIII Brazilian Symposium on, 2015, pp. 189–198. `doi:10.1109/SBRC.2015.31`.

[3] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, P. Levis, Collection tree protocol, in: Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems, SenSys '09, 2009, pp. 1–14.

[4] T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, J. Vasseur, R. Alexander, RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks, RFC 6550 (Proposed Standard) (2012).

[5] S. Duquennoy, O. Landsiedel, T. Voigt, Let the tree bloom: Scalable opportunistic routing with orpl, in: Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems, SenSys '13, ACM, New York, NY, USA, 2013, pp. 2:1–2:14. `doi:10.1145/2517351.2517369`.
URL `http://doi.acm.org/10.1145/2517351.2517369`

[6] A. Reinhardt, O. Morar, S. Santini, S. Zoller, R. Steinmetz, Cbfr: Bloom filter routing with gradual forgetting for tree-structured wireless sensor networks with mobile nodes, in: World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2012 IEEE International Symposium on a, 2012, pp. 1–9. `doi:10.1109/WoWMoM.2012.6263685`.

[7] B. P. Santos, M. A. Vieira, L. F. Vieira, eXtend collection tree protocol, in: Wireless Communications and Networking Conference (WCNC), 2015 IEEE, 2015, pp. 1512–1517. `doi:10.1109/WCNC.2015.7127692`.

[8] P. Levis, N. Patel, D. Culler, S. Shenker, Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks, in: Proceedings of the 1st Conference on Symposium on Networked Systems Design and Implementation - Volume 1, NSDI'04, 2004, pp. 2–2.

[9] B. N. Clark, C. J. Colbourn, D. S. Johnson, Unit disk graphs, Discrete Math. 86 (1-3) (1991) 165–177.

[10] Levis, Philip and Madden, Sam and Polastre, Joseph and Szewczyk, Robert and Whitehouse, Kamin and Woo, Alec and Gay, David and Hill, Jason and Welsh, Matt and Brewer, Eric and others, TinyOS: An operating system for sensor networks, Ambient intelligence 35 (2005) 115–148.

[11] P. Levis, N. Lee, M. Welsh, D. Culler, Tossim: Accurate and scalable simulation of entire tinyos applications, in: Proceedings of the 1st International Conference on Embedded Networked Sensor Systems, SenSys '03, ACM, New York, NY, USA, 2003, pp. 126–137. `doi:10.1145/958491.958506`. URL `http://doi.acm.org/10.1145/958491.958506`

[12] A. Dunkels, B. Gronvall, T. Voigt, Contiki - a lightweight and flexible operating system for tiny networked sensors, in: IEEE LCN, IEEE Computer Society, Washington, DC, USA, 2004, pp. 455–462.

[13] J. Eriksson, F. Österlind, N. Finne, N. Tsiftes, A. Dunkels, T. Voigt, R. Sauter, P. J. Marrón, Cooja/mspsim: Interoperability testing for wireless sensor networks, in: Proceedings of the 2Nd International Conference on Simulation Tools and Techniques, Simutools'09, 2009, pp. 27:1–27:7.

[14] N. Baccour, A. Koubâa, L. Mottola, M. A. Zúñiga, H. Youssef, C. A. Boano, M. Alves, Radio link quality estimation in wireless sensor networks: A survey, ACM Trans. Sen. Netw. 8 (4) (2012) 34:1–34:33.

[15] M. Bezunartea, M. Gamallo, J. Tiberghien, K. Steenhaut, How interactions between rpl and radio duty cycling protocols affect qos in wireless sensor networks, in: Proceedings of the 12th ACM Symposium on QoS and Security for Wireless and Mobile Networks, Q2SWinet '16, ACM, New York, NY, USA, 2016, pp. 135–138. `doi:10.1145/2988272.2988279`. URL `http://doi.acm.org/10.1145/2988272.2988279`

[16] C. Perkins, E. Belding-Royer, S. Das, Ad hoc on demand distance vector (AODV) routing (RFC 3561), IETF MANET Working Group.

[17] D. Johnson, Y. Hu, D. Maltz, The dynamic source routing protocol (DSR) for mobile ad hoc networks for IPV4, RFC: 4728.

[18] N. d. S. R. Júnior, M. A. Vieira, L. F. Vieira, O. Gnawali, Codedrip: Data dissemination protocol with network coding for wireless sensor networks, in: Wireless Sensor Networks, Springer, 2014, pp. 34–49.

[19] Dawson-Haggerty, Stephen and Tavakoli, Arsalan and Culler, David, Hydro: A hybrid routing protocol for low-power and lossy networks, in: Smart Grid Communications (SmartGridComm), 2010 First IEEE International Conference on, IEEE, 2010, pp. 268–273.

[20] U. Palani, V. Alamelumangai, A. Nachiappan, Hybrid routing and load balancing protocol for wireless sensor network, Wireless Networks (2015) 1–8`doi:10.1007/s11276-015-1110-1`.
URL `http://dx.doi.org/10.1007/s11276-015-1110-1`

[21] M. N. Moghadam, H. Taheri, M. Karrari, Multi-class multipath routing protocol for low power wireless networks with heuristic optimal load distribution, Wirel. Pers. Commun. 82 (2) (2015) 861–881. `doi:10.1007/s11277-014-2257-2`.
URL `http://dx.doi.org/10.1007/s11277-014-2257-2`

[22] M. A. Lodhi, A. Rehman, M. M. Khan, F. B. Hussain, Multiple path rpl for low power lossy networks, in: Wireless and Mobile (APWiMob), 2015 IEEE Asia Pacific Conference on, 2015, pp. 279–284. `doi:10.1109/APWiMob.2015.7374975`.

[23] E. C. Perkins, D. Johnson, J. Arkko, Mobility support in ipv6, Tech. rep., RFC 6275 (2011).